Asynchronous Consensus with Bounded Memory

Carole Delporte-Gallet, Hugues Fauconnier

Asynchronous Consensus with Bounded Memory

Carole Delporte-Gallet[1] and Hugues Fauconnier[1]

U. Paris-Diderot, France. {cd,hf}@liafa.univ-paris-diderot.fr*

Abstract. We present here a bounded memory consensus Obstruction-Free algorithm for the asynchronous shared memory model. More precisely for a set of n processes, this algorithm uses $n+1$ multi-writer multi-reader (MWMR) registers, each of these registers being of size $O(\log(n))$ bits. Then we get a $O(n\log(n))$-bits size complexity consensus algorithm with single-writer multi-reader (SWMR) registers and a $O(n\log(n))$-bits complexity consensus algorithm in the asynchronous message passing model with a majority of correct processes. As it is easy to ensure the Obstruction-Free assumption with randomization (or with leader election failure detector Ω) we obtain a bounded memory size randomized consensus algorithm and a bounded memory size consensus algorithm with failure detector.

Keywords: Shared memory, space complexity, consensus

1 Introduction

Because of its practical impact and for theoretical reasons, the consensus problem is one of the most interesting problem in fault-tolerant computing. Recall that in the consensus problem each process begins with an initial value and has to decide one value (*termination*), this decided value has to be an initial value of some process (*validity*) and all processes have to decide the same value (*agreement*). But in message passing or shared memory asynchronous systems there is no deterministic solution for the consensus if at least one process may crash [16]. To circumvent this negative result, several ways have been proposed. One of them is to consider *randomized consensus*: with randomization, it is possible to ensure the safety of the consensus (agreement and validity) and the liveness property (termination) with a probability equal to 1 [6, 2]. Another way is to add *failure detectors* [9] to the system. Failure detectors are distributed oracles that give processes information about failures. In this way it has been proved [8] that Ω, a failure that ensures that eventually all correct processes agree on the id of the same correct processes (a leader), is a weakest failure detector to solve the consensus problem (i.e. if another failure detector enables to solve the consensus, then Ω may be implemented from the information given by that failure detector).

In shared memory models, if we limit the concurrency in such a way that each correct process is running alone for an arbitrary long time consensus becomes solvable. An *Obstruction-Free* algorithm [20, 15] is an algorithm ensuring safety in all cases and termination under that concurrency property. More precisely it is a deterministic algorithm that guarantees that any process will decide if it performs enough steps alone (i.e. without

* Supported by ANR DISPLEXITY.

interference with other processes). Randomization [18] or failure detector Ω enable (with bounded memory size) to ensure that all correct processes are running alone for an arbitrary long times and then an Obstruction-Free algorithm gives directly a randomized algorithm or a deterministic algorithm with failure detector Ω. Hence in the following we consider Obstruction-Free consensus algorithms.

The main result of the paper is to present an Obstruction-Free consensus algorithm tolerating any number of process crashes in the shared memory model using with a bounded size of memory. For a system of n processes, the algorithm uses only $n+2$ registers that may be written or read by all processes (multi-writer multi-reader MWMR registers). Moreover, each of these registers contains at most $O(\log(n))$ bits. In fact our algorithm uses atomic snapshot [1] to access a memory of $n+1$ registers. In [18], it has been proved that atomic snapshot of registers can be Obstruction-Free implemented with one additional atomic register of size $O(\log n)$-bits. Hence with MWMR registers (and only read-write operations) the size of the memory for our algorithm is bounded by $O(n \log(n))$.

Following classical results (e.g. [31, 26, 28]) one MWMR register shared by n processes may be implemented with $O(n^2)$ single-writer multi-reader (SWMR) registers and bounded timestamps [24, 14]. Hence we get a bounded memory size Obstruction-Free consensus algorithm with SWMR registers. In this way, with any number of crashes we obtain a randomized consensus algorithm with bounded memory size and a consensus algorithm with bounded memory size and failure detector Ω.

SWMR registers may be implemented in message passing models with a majority of correct processes using bounded timestamps [3]. Then if each register to be implemented has a bounded size, we obtain a bounded size of memory for the exchanged messages in the implementation. Hence in asynchronous message passing models and a majority of correct processes we obtain a randomized consensus algorithm with bounded memory size and a consensus algorithm with bounded memory size using failure detector Ω. Without a majority of correct processes, registers can be implemented with bounded memory size using failure detector Σ [13] then we obtain also a consensus algorithm with bounded memory size using failure detector $\Sigma \times \Omega$ that tolerates any number of crashes.

Most of consensus algorithms work with rounds that are identified by counters that are (unbounded) integers (e.g. [9, 25, 23, 30, 23]) to the best of our knowledge our algorithm is the first one working with bounded memory size.

We also present similar results concerning the k set-agreement. In this problem processes have to decide on at most k values (consensus corresponds to 1-set agreement).

2 Model

The model we consider is the standard asynchronous shared-memory model of computation with $n \geq 2$ processes [1] communicating by reading and writing to a fixed set of shared registers [5, 22]. The processes have unique ids in $\{1, \ldots, n\}$. Processes may take a finite (in this case the process is faulty) or an infinite number of steps (in this case the process is correct) and we assume that at least one process is correct.

[1] The number n of processes is given and is then considered as a constant.

Shared memory. The shared memory consists of a set of atomic Multi-Writer Multi-Readers (MWMR) registers. All processes can read and write any MWMR register and these operations are atomic or linearizable [19]. For short, we usually omit the term atomic. A process executes its code by taking three types of atomic steps: the *read* of a register, the *write* of a register, and the modification of its local state. *Atomic snapshots* [1] is another way to access memory. Atomic snapshot is defined by two more powerful operations: *update* and *scan* [1] on an array of m MWMR registers. An update operation takes a register and a data value as arguments and does not return a value. It writes the data value in the register. The scan operation has no argument and returns an array of m elements. The returned array is a *snapshot* of the memory, that is an instantaneous view of the registers.

Space complexity. We consider here the space complexity of algorithms. By space complexity we mean the maximum over all the runs of the algorithm of the sum of all the sizes of all shared registers. An bounded memory algorithm is an algorithm such that there exists a constant B, such that in any run of the algorithm the sum over all shared MWMR registers of the size of registers is less than B.

Obstruction Freedom. The *Obstruction-Free* [20] progress condition states that when a process takes enough steps alone it must terminate its task. In the following we consider deterministic Obstruction-Free implementations.

Giakkoupis et al. give in [18] an Obstruction-Free linearizable implementation of atomic snapshot that uses, in addition to the array of m MWMR registers, say R, one MWMR registers of size $O(\log(n))$. Note that in this implementation, an update operation writes in the register the id of the writer, so if the value written in R does not yet contain the id of the writer, the space complexity is augmented by a multiplicative factor $\log(n)$.

In [1], it is observed that if every write leaves a unique indelible mark whenever it is executed in R, then if two consecutive reads of R return identical values then the values returned constitute a snapshot. Using this idea, [18] designed an Obstruction-Free linearizable implementation of scan and update. It uses an additional register S. To perform its j-th update of some value x in register $R[i]$, p first writes its id in S and then it writes the triple $(x, p, j \bmod 2)$ in $R[i]$. To execute a $scan()$, process p first writes its id in S. Then it performs two consecutive reads of all the registers of R: r and r'. Finally, the process reads S. If S does not contain p's id or if the two consecutive views r and r' are not equal then p starts its $scan()$ over; otherwise it returns view r.

Proposition 1 ([18]). *Assuming that each written value contains the id of the writer and that the size of the written value is w, there is an Obstruction-Free linearizable implementation of scan/update of m MWMR registers with one MWMR register of size $O(\log(n))$ and m MWMR registers of size $w + 1$.*

We consider the classical consensus decision task [10] in which each process proposes its input value and has to irrevocably decide on one of the proposed inputs, such that there is at most one decided value. We assume that the input values come from a finite set of values *Values*. Consensus is defined by three requirements:

– *agreement:* at most one value is decided,

– *validity:* if a process decides value v, v is the input value of some process,
– *termination:* if a process takes an infinite number of steps then it decides.

It is well known that consensus is not Wait-Free solvable.[2] Moreover consensus is not solvable as soon as at least one process is faulty [16]. It is also well known that there exits Obstruction-Free consensus implementations (see for example [4, 11, 12]). So we study the Obstruction-Free solvability of consensus. We are interested here by the space complexity of the implementations and we are going to prove the existence of bounded memory consensus implementations.

3 Algorithm

Algorithm of Figure 1 solves Obstruction-Free consensus with $n+2$ MWMR registers.

In this algorithm the processes share $n+1$ registers and use these registers by snapshots. The implementation of scan/update is the Obstruction-Free implementation described in [18]. As previously mentioned, this implementation uses one additional MWMR register.

Each process p maintains a variable *prop*, its proposal. Initially this variable contains its input value v_p. Each process repeatedly takes a scan of the registers, sets its variable *prop* to v if it finds two registers with the same content (v, q) for some q, and updates some register with a pair formed by *prop* and its id. It decides when it finds in each register this pair. To avoid that a process alternates between two proposals, a process keeps its proposal *prop* if the pair composed with its proposal *prop* and some id appears twice.

Then, in an Obstruction-Free run, the process that takes enough steps alone updates all registers with its proposal and decides.

The agreement property is more intricate. If a process p decides a value d, then (d, p) is written in $(n+1)$ registers. We argue that forever at least two registers contain the pair (d, q) for some q and for any (d', q') with $d \neq d'$,(d', q') is contained in at most one register. Intuitively, consider the first scan made after the decision and q the process that made this scan. As processes repeatedly perform a scan and then an update, at most $n-1$ processes may have written a proposal different from the decided value d. And so it remains at least two registers with (d, p) and the other registers contains either (d, p) or the value written by processes since the decision. So after this scan, the value d is adopted by q.

Here we use a trick, when a process changes its proposal it updates the *same* register. So if q writes in register $R[i]$ and q has changed its proposal, the next update of q will be on the same register $R[i]$. After this second update, q performs a scan and this time, it does not modify its proposal. It will update some other register with (d, q). Perhaps it updates one of the two registers that contains (d, p) but in this case we have two registers with the value (d, q). It is also possible that we have (d, p) in at least two registers and (d, q) in two registers. But the main point is that we keep the property that for any (d', x) with $d' \neq d$, the pair (d', x) is contained in at most one register and (d, x) for some x is in two registers. The agreement works in the same way if q does not change its proposal after its first scan (q has already adopted d as proposal).

[2] The *wait-free* [19] progress condition states that each process terminates its operations in finite number of its own steps or equivalently with any number of failures

```
Shared variables:
    R :array of R[0,..,n] of pairs (value, process_ID) initialized to (⊥, ⊥)
CODE FOR PROCESS p
array of  r[0,..,n] of pairs $(value, process\_ID)$
$prop = v_p$                                                          /*$p$ proposal */
$pos = 0$
forever do
    r[0,..,n]= $R.scan()$
    if $\forall i\ r[i] = (prop, p)$ then decide $prop$ ; exit                /*$p$ decision*/
    if $\exists i, j : (r[i] = r[j] \wedge r[i] \neq (\perp, \perp) \wedge r[i].value \neq prop)$
        $\wedge \nexists i, j : r[i] = r[j] \wedge (r[i].value = prop)$
    then        /* two identical values in $R$ different from the current $p$ proposed value */
                                                  /*keep the same register to write */
        let $v$ be $r[i].value$ such that $\exists j : r[i] = r[j] \wedge (r[i] \neq (\perp, \perp)) \wedge r[i].value \neq prop$
        $prop = v$
    else                    /*keep the same proposal, chose another register to write r */
        let $k$ be the smallest index such that $r[k] \neq (prop, p)$
        $pos = k$
    $R.update(pos, (prop, p))$
```

Fig. 1. Consensus with $n + 1$ MWMR registers.

Now we proceed with the correctness proof of the algorithm Figure 1.

By definition scan and update are linearizable and when we say that some scan or update operation *op* occurs at some time τ, time τ is the linearization time of this operation.

We first prove the termination property of the algorithm:

Proposition 2. *In Obstruction-Free run, if some process p takes steps for an arbitrary long time, then p decides.*

Proof. With obstruction freedom, without loss of generality, assume that a process, say p, is eventually the only process taking steps. Then there is a time τ after which no other process takes steps. Notice that in particular, after time τ, only p may modify registers in R.

Consider the first update by p after time τ, p writes $(prop, p)$ in $R[pos]$. After this update, p scans the registers.

There are two cases:

- p changes its proposal Line 8 by $prop'$. It has found i and j such that $r[i] = r[j] \wedge r[i] \neq (\perp, \perp) \wedge r[i].value \neq prop$. Note that as it takes steps alone we have $i \neq pos$ and $j \neq pos$. In its next scan, it finds again $r[i] = r[j] \wedge r[i].value = prop'$ and it keeps its proposal and updates a register $R[pos']$ for some $pos' \neq pos$. Then forever there are at least two registers with $(prop', p)$ in R and p never changes its proposal. Then p updates the registers one by one until all registers contain $(prop', p)$ and then it decides.

– p does not change its proposal Line 11. In this case it has found either $\nexists i, j : (r[i] = r[j] \wedge r[i] \neq (\perp, \perp) \wedge r[i].value \neq prop)$ or $\exists i, j : r[i] = r[j] \wedge (r[i].value = prop)$. There are two cases:
  - If $\exists i, j : r[i] = r[j] \wedge (r[i].value = prop)$, then after its next scan, p keeps also its proposal and updates a register $R[pos']$ for some $pos' \neq pos$. Then forever there are at least two registers with $(prop, p)$ in R and p never changes its proposal. It updates the registers one by one until all registers contain $(prop, p)$ and then it decides.
  - Else $\nexists i, j : (r[i] = r[j] \wedge r[i] \neq (\perp, \perp) \wedge r[i].value \neq prop)$ and $\nexists i, j : r[i] = r[j] \wedge (r[i].value = prop)$. So p updates its variable pos to pos' with $pos \neq pos'$ Line 13. After the next update, in all the scans made by p there are at least two registers that contain $(prop, p)$. Then p updates the registers one by one until all registers contain $(prop, p)$ and then it decides.

We now proceed to prove the agreement and the validity properties:

Proposition 3. *If a process decides, it decides the input value of some processes.*

Proof. The decided value is the first argument of a pair that is written in all registers. From the algorithm, the proposal of some process, and consequently the value written in registers, is either an input value or an update made Line 10 i.e. some $r[i].value$ such that $\exists j : r[i] = r[j] \wedge (r[i] \neq (\perp, \perp)) \wedge r[i].value \neq prop$. Then, by induction, the decided value is the input of some processes.

Proposition 4. *If two processes decide, they decide the same value.*

Proof. Consider the first process, say p, that is ready to decide d and the time τ at which it has found a scan with (d, p) written in $n+1$ registers. After time τ, p never writes in the shared memory and so at most $n-1$ processes writes in $n+1$ registers.

We argue that each time after τ a process scans the memory it finds (1) at least two registers with a pair (d, q) for some q, and (2) any pair (d', x) with $d' \neq d$ and x a process in at most one register.

At any time $t > \tau$, let X^t let the set of processes that have made at least one write between τ and t. We have $n-1 \geq |X^t|$. Let Y^t be the set of registers that contains (d', x) with $d' \neq d$ and x in X at time t. Let $\tau' > \tau$ such that some process, say q, makes for the first time a second write since τ. At any time t between τ and τ', we have $|X^t| \geq |Y^t|$.

Until τ', there are at most $n-1$ values in X^t and in Y^t and properties (1) and (2) are satisfied.

If q makes its second write at τ', it has made its scan between τ and τ'. As (1) and (2) are satisfied, the proposal of q after this scan will be d. There are two cases:

Case 1: After this scan, q has modified its proposal. Then q writes (d, q) in the same position. If $|Y^{\tau'}| < |Y^{\tau'-1}|$ then $|X^{\tau'}| \geq |Y^{\tau'}| + 1$ and properties (1) and (2) are satisfied. If $|Y^{\tau'}| = |Y^{\tau'-1}|$, this means that another process, say w, has already written (d, w) in the same place then again $|X^{\tau'}| \geq |Y^{\tau'}| + 1$ and properties (1) and (2) are satisfied.

Case 2: After this scan, q has not modified its proposal. Then its proposal was already d, q has written (d, q) for its first write in some index i, consequently i is not in $Y^{\tau'}$. Then again $|X^{\tau'}| \geq |Y^{\tau'}| + 1$. As $n-1 \geq |X^{\tau'}|$, it remains 3 registers with either (d, p) or (d, q)

then one of these pairs is in two registers and (1) is ensured. As q writes (d, q), (2) remains true.

We consider the time τ'' of a next write of a process such that it is its second write or its third write after τ. Between τ' and τ'' it may happen that some process has made its first write but in this case X increases as least as Y and we keep that at any time t, $|X^t| \geq |Y^t| + 1$ before this write.

If it is the second write of some process, we have the same argument as previously. (1) and (2) holds and $|X^{\tau''}| \geq |Y^{\tau''}| + 2$. If it is the third write of some process then before this write, it remains 3 registers with either (d, p) or (d, q). If this write is in one of these registers the property (1) is always ensured. If it is in another register then the property (1) trivially holds. As q writes (d, q), (2) remains true.

The proof is made iteratively until time σ at which all processes that take steps in the run have already made at least two writes. As in its second write (and all its next writes) a process k writes (d, k), the property (2) holds. From time σ, as $n-1$ processes write in $n+1$ registers, there is a process, say k, that writes twice its pair (d, k) and then the property (1) holds. After σ, all processes have d as proposals and never change their proposals.

Theorem 1. *With the Obstruction-Free implementation of snapshot in [18], the algorithm of Figure 1 solves Obstruction-Free consensus in space memory $O(n \log n)$. More precisely, it uses $n + 2$ MWMR registers of size $O(\log n)$.*

4 Applications

4.1 Randomized algorithm

A randomized implementation of consensus that tolerates any number of crashes ensures the agreement and the validity in all runs and the termination (each correct process decides) with probability one.

[18] shows that if there is a deterministic Obstruction-Free algorithm which guarantees that any process finishes after it has executed at most b steps, for some constant b, without interference from other processes then the algorithm can be transformed into a randomized one that has the same space complexity. Observe that in our Obstruction-Free implementation, if a process p takes steps alone, it terminates after $(n + 2)$ scan and update operations. It may happen that when it takes steps alone it is in a middle of its scan/update. It needs $(n+1)$ complete scan/update alone to terminate. For each $scan()$, p reads twice the array R and the additionnel register then if p takes steps alone it decides in at most $c(n + 2)(n + 1)$ atomic steps for some constant c. Then from [18] and Theorem 1:

Corollary 1. *There is a randomized implementation of consensus that tolerates any number of crashes with MWMR in space memory $O(n \log(n)))$. More precisely, it uses $n+2$ MWMR registers of size $O(\log(n))$.*

4.2 SWMR registers

Single-Writer Multi-Reader registers are considered as more primitive than Multi-Writer Multi-Reader registers. Then we consider the space complexity for SWMR registers. Here, space complexity is again defined as the maximum over all the runs of the algorithms of the sum of the sizes of all shared *SWMR* registers.

Following classical results (e.g. [31, 26, 28]) one MWMR registers may be implemented with $O(n^2)$ single-writer multi-reader (SWMR) registers and bounded timestamps [24, 14]. Hence with these implementations we get again a bounded size memory Obstruction-Free deterministic consensus algorithm and a randomized consensus algorithm that tolerates any number of crashes with SWMR registers. Therefore:

Corollary 2. *There is an Obstruction-Free consensus deterministic algorithm and a randomized consensus algorithm that tolerates any number of crashes with SWMR registers in bounded memory.*

4.3 Failure detector

A failure detector [9] outputs at each process some hints about the failure pattern of the processes. Failure detectors enable to solve consensus. In particular, failure detector Ω [8] that eventually outputs to each process the same correct process id, the eventual leader, is the weakest failure detector to achieve consensus in shared memory. That means that from the output of any failure detector enabling to solve consensus it is possible to implement Ω.

In asynchronous shared memory model augmented with the leader election failure detector Ω, we also get a bounded memory implementation of consensus.

Indeed assuming a failure detector Ω, if a process takes steps only when it considers itself as the leader, the Ω properties imply that eventually only one process, the eventual correct leader, will take steps. In this way, we get an emulation of the Obstruction-Free property.

Running a consensus Obstruction-Free algorithm in this way with Ω, gives an algorithm with failure detector Ω in which the leader eventually decides. Details are given in Figure 2. To ensure that the other correct processes decide, when the leader decides it writes the decided value in a register, and the other processes adopt this value as their decision value.

Hence contrary to classical consensus algorithms with failure detector Ω that use generally variables like counters of "rounds" (e.g. [27, 11]) that take unbounded values, we obtain a bounded memory consensus algorithm with failure detector Ω.

Thus, with Theorem 1:

Corollary 3. *There is a deterministic consensus algorithm in shared memory model MWMR registers augmented with failure detector Ω with space complexity $O(n\log(n))$. There is a deterministic consensus algorithm in shared memory model SWMR registers augmented with failure detector Ω with bounded memory.*

4.4 Message passing

Consider now the message passing asynchronous model. In this model processes communicate by messages. We assume here that the communication is reliable (no loss, no corruption, no duplication). SWMR registers may be implemented in message passing system with a majority of correct processes [3] using bounded timestamp and hence with bounded size memory.

```
Shared variables:
        $DEC : Values$ initialized to $\bot$
CODE FOR PROCESS $p$

while $true$ do
    if $(\Omega = p)$                      /* $\Omega$ outputs id of the supposed leader*/
        then
            run the next step of an Obstruction-Free consensus algorithm
            if $p$ decides $v$ in this step then $DEC = v$; $decide(DEC)$; exit
        else
            if $DEC \neq \bot$ then $decide(DEC)$; exit
```

Fig. 2. Running Obstruction-Free algorithm with Ω.

Without a majority of correct processes, SWMR registers can be implemented using the failure detector Σ [13]. Moreover in way similar to [3], this implementation can be made with bounded size memory. Failure detector Σ outputs at each process a list of processes that eventually contains only correct processes, such that each output has a non empty intersection with any other output. Note that $\Sigma \times \Omega$ is the weakest failure detector to solve the consensus with any number of crashes [13].

Corollary 4. *Assuming a majority of correct processes, there is a deterministic consensus algorithm in asynchronous message passing systems with failure detector Ω with bounded memory.*

Assuming any number of crashes, there is a deterministic consensus algorithm in asynchronous message passing systems with failure detector $\Sigma \times \Omega$ with bounded memory.

Assuming a majority of correct processes, there is a randomized algorithm in asynchronous message passing with bounded memory.

5 Extensions

We consider the classical *k-set agreement* decision task [10] in which each process proposes its input value and has to decide on one of the input values, such that there are at most k decided values. We assume that the input values come from a finite set of values *Values*. There are three requirements: at most k values are decided (k-*agreement*), if a process decides v, this value is the input of some process (*validity*), and if a process takes an infinite number of steps then it decides (*termination*). When $k = n - 1$, k-set agreement is also known as set agreement. The *consensus* is nothing else than 1-set agreement.

It is well known that k-set agreement is not wait-free solvable and even, it is not solvable with k faulty processes [7, 21, 29] in shared memory, so we study the Obstruction-Free solvability of k-set agreement. It is also known that there exits an Obstruction-Free k-set agreement implementation (for example [4, 12]), we are interested here on the space complexity of an implementation.

From our algorithm Figure 1, we can derive an Obstruction-Free implementation of k-set agreement in bounded memory. More precisely, using scan/update the algorithm uses $n-k+2$ MWMR registers of size $O(\log(n))$ and we need one additional register of size $O(\log(n))$ to implement Obstruction-Free the scan/update operations.

The principles of the algorithm is the same as those of the consensus algorithm Figure 1, but instead of $n+1$ registers we have only $n-k+2$ registers. We observe that if we consider the last $n-k+1$ processes ready to decide, it is possible that the other $k-1$ processes have decided some value, but these $n-k+1$ processes share now a set of $(n-k+1)+1$ registers and are, roughly speaking, in the condition in which they can achieve consensus as in the consensus algorithm Figure 1. So there is at most k decided value.

The algorithm is given Figure 3 and its proof is similar to the proof of the consensus algorithm.

```
Shared variables:
    R :array of R[0,..,n-k+1] of pairs (value, process_ID) initialized to (⊥,⊥)
CODE FOR PROCESS p
array of  r[0,..,n-k+1] of pairs (value, process_ID)
prop = v_p                                                    /*p proposal */
pos = 0
forever do
    r[0,..,n-k+1]= R.scan()
    if ∀i r[i] = (prop, p) then decide prop ; exit            /*p decision*/
    if ∃i,j : (r[i] = r[j] ∧ r[i] ≠ (⊥,⊥) ∧ r[i].value ≠ prop)
        ∧∄i,j : r[i] = r[j] ∧ (r[i].value = prop)
    then        /* two identical values in R different from the current p proposed value */
                                              /*keep the same register to write */
        let v be r[i].value such that ∃j : r[i] = r[j] ∧ (r[i] ≠ (⊥,⊥)) ∧ r[i].value ≠ prop
        prop = v
    else                 /*keep the same proposal, chose another register to write r */
        let k be the smallest index such that r[k] ≠ (prop, p)
        pos = k
    R.update(pos, (prop, p))
```

Fig. 3. k-set agreement with $n-k+2$ MWMR registers.

If we combined again the known results we get:

Theorem 2. *There is an Obstruction-Free deterministic k-set agreement algorithm and randomized k-set agreement algorithm that tolerates any number of fault with MWMR registers in space complexity* $O((n-k)\log(n))$. *More precisely, it uses* $n-k+3$ *MWMR registers of size* $O(\log(n))$.

Theorem 3. *There is an Obstruction-Free deterministic k-set agreement algorithm and a randomized k-set agreement algorithm that tolerates any number of fault with SWMR registers in bounded memory.*

The weakest failure detector to achieve k-set agreement in shared memory is $vector\Omega_k$ [17]. This failure detector outputs at each process a vector of size k, such that there exists some index i, such that eventually at each process the i-th index of the output is the same correct process. A traditional implementation of k-set agreement with this failure detector is the following: Each process runs in parallel k instances of a consensus algorithm with a failure detector such that this algorithm achieves termination with Ω and agreement and validity with any failure detector. Note that our algorithm Figure 2 satisfies these properties. When a process decides in one of its instances of consensus, it decides for k-set agreement. As one index is an Ω at least one instance decides for all processes. As all instances of consensus algorithm achieve agreement and validity there is at most k decisions and each decision is the initial value of some process.

Theorem 4. *There is a deterministic k-set agreement algorithm in shared memory model MWMR registers augmented with failure detector $vector\Omega$ in space complexity $O(kn\log(n))$. There is a deterministic k-set agreement algorithm in shared memory model SWMR registers augmented with failure detector $vector\Omega$ in bounded memory.*

Finally with the result of [18], we get:

Theorem 5. *There is a randomized k-set agreement algorithm in shared memory model MWMR registers in space complexity $O(kn\log(n))$. There is a deterministic k-set agreement algorithm in shared memory model SWMR registers in bounded memory.*

References

1. Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.
2. James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.
3. Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message passing systems. *Journal of the ACM*, 42(2):124–142, January 1995.
4. Hagit Attiya, Rachid Guerraoui, Danny Hendler, and Petr Kuznetsov. The complexity of obstruction-free implementations. *J. ACM*, 56(4), 2009.
5. Hagit Attiya and Jennifer Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics.* John Wiley & Sons, 2004.
6. Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC '83: Proceedings of the annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.
7. Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100. ACM Press, 1993.
8. Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.
9. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

10. Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.
11. Carole Delporte-Gallet and Hugues Fauconnier. Two consensus algorithms with atomic registers and failure detector omega. In *Distributed Computing and Networking, 10th International Conference, ICDCN 2009*, volume 5408 of *LNCS*, pages 251–262. Springer, 2009.
12. Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Sergio Rajsbaum. Black art: Obstruction-free k-set agreement with $|mwmr \text{ registers}| < |processes|$. In *NETYS*, volume 7853 of *LNCS*, pages 28–41, 2013.
13. Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Tight failure detection bounds on atomic object implementations. *Journal of the ACM*, 57(4), 2010.
14. Danny Dolev and Nir Shavit. Bounded concurrent time-stamping. *SIAM J. Comput.*, 26(2):418–455, 1997.
15. Faith Ellen Fich, Victor Luchangco, Mark Moir, and Nir Shavit. Obstruction-free algorithms can be practically wait-free. In *Proceedings of the International Symposium on Distributed Computing*, pages 493–494, 2005.
16. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
17. Eli Gafni and Petr Kuznetsov. On set consensus numbers. *Distributed Computing*, 24(3-4):149–163, 2011.
18. George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. An o(sqrt n) space bound for obstruction-free leader election. In *DISC*, volume 8205 of *LNCS*, pages 46–60. Springer, 2013.
19. Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):123–149, January 1991.
20. Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *ICDCS*, pages 522–529. IEEE Computer Society, 2003.
21. Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(2):858–923, 1999.
22. Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2008.
23. Michel Hurfin and Michel Raynal. A simple and fast asynchronous consensus based on a weak failure detector. *Distributed Computing*, 12(4), 1999.
24. A. Israeli and M. Li. Bounded time-stamps. *Distributed Computing*, 6(4), July 1993.
25. Leslie Lamport. The Part-Time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.
26. Ming Li, John Tromp, and Paul M. B. Vitányi. How to share concurrent wait-free variables. *J. ACM*, 43(4):723–746, 1996.
27. A. Mostéfaoui and M. Raynal. Leader-based consensus. *Parallel Processing Letters*, 11(1):95–107, 2001.
28. Michel Raynal. *Concurrent Programming - Algorithms, Principles, and Foundations.* Springer, 2013.
29. Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. on Computing*, 29:1449–1483, 2000.
30. André Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, 1997.
31. Ambuj K. Singh, James H. Anderson, and Mohamed G. Gouda. The elusive atomic register. *J. ACM*, 41(2):311–339, 1994.